Amendments to the specification,

Marked version of the replacement paragraph(s)/section(s), pursuant to 37 CFR 1.121(b)(1)(ii):

Please replace paragraph [0003] with the following rewritten paragraph:

Object Description: SourceCode.txt, size: 4KB, created: 02/03/2004 1:14PM; Object ID: File No. 1; Object Contents: Source Code.

Object Description: SampleApp.txt, size: 15508 Bytes, created: 05/23/07 2:27:24 PM; Object ID: File No. 2; Object Contents: Source code.

Please replace paragraph [0008] with the following rewritten paragraph:

Owing to their digital nature, computers essentially only understand "machine code," i.e., the low-level, minute instructions for performing specific tasks -- the sequence of ones and zeros that are interpreted as specific instructions by the computer's microprocessor. Since machine language or machine code is the only language computers actually understand, all other programming languages represent ways of structuring human language so that humans can get computers to perform specific tasks. While it is possible for humans to compose meaningful programs in machine code, practically all software development today employs one or more of the available programming languages. The most widely used programming languages are the "high-level" languages, such as C, C++, Pascal, or more recently Java® and C#. These languages allow data structures and algorithms to be expressed in a style of writing that is easily read and understood by fellow programmers.

Please replace paragraph [0009] with the following rewritten paragraph:

A program called a "compiler" translates these instructions into the requisite machine language. In the context of this translation, the program written in the high-level language is called the "source code" or source program. The ultimate output of the compiler is a compiled module such as a compiled C "object module", which includes instructions for execution ultimately by a target processor, or a compiled Java® class, which includes bytecodes for execution ultimately by a Java® virtual machine. A Java® compiler generates platform-neutral "bytecodes", an architecturally neutral, intermediate format designed for deploying application code efficiently to multiple platforms.

Please replace paragraph [0010] with the following rewritten paragraph:

"Visual" development environments, such as Borland's JBuilder® or Borland's C# Builder(TM), are the preferred application development environments for quickly creating production applications. Such environments are characterized by an integrated development environment (IDE) providing a form painter, a property getter/setter manager ("inspector"), a project manager, a tool palette (with objects which the user can drag and drop on forms), an editor, and a compiler. In general operation, the user "paints" objects on one or more forms, using the form painter. Attributes and properties of the objects on the forms can be modified using the property manager or inspector. In conjunction with this operation, the user attaches or associates program code with particular objects on screen (e.g., button objects); the editor is used to edit program code which has been attached to particular objects. After the program code has been developed, the compiler is used to generate binary code for execution on a machine (e.g., Intel native machine code for execution on Intel-compatible microprocessors or Java® bytecode for execution on a Java® virtual machine).

Please replace paragraph [0012] with the following rewritten paragraph:

Various approaches have been made to assist users in handling this increased complexity. A current technology for understanding complex object-oriented software systems is the Unified Modeling Language (UML). UML is a standard graphical language for modeling object-oriented systems. For further information on UML, see e.g., "OMG Unified Modeling Language Specification (Version 1.5, March 2003)" available from the Object Management Group, Inc., the disclosure of which is hereby incorporated by reference. A copy of this specification is available via the Internet (e.g., currently at (www)~~www.~~omg.org). UML, at its simplest, is a language that graphically describes a set of elements. At its most complex, UML is used to specify, visualize, construct, and document not only software systems but also business models and non-software systems. Much like a blueprint for constructing a building, UML provides a graphical representation of a system design that may be used by developers to assure architectural soundness of a system. UML is frequently used to design the relationships between components (e.g., classes in the case of a Java program) before such components are developed and implemented in source code format. For example, UML class diagrams allow the structure of a system to be specified in a language-neutral fashion, without involvement in all of the various implementation details such as persistence, business domain implementation, and technical behavior of the system.

Please replace paragraph [0033] with the following rewritten paragraph:

UML: UML standards for the Unified Modeling Language, a standard graphical language for modeling object-oriented systems. For further information on UML, see e.g., "OMG Unified Modeling Language Specification (Version 1.5, March 2003)" available from the Object Management Group, Inc., the disclosure of which is hereby incorporated by reference. A copy of this specification is available via the Internet (e.g., currently at (www)~~www.~~omg.org).

Please replace paragraph [0034] with the following rewritten paragraph:

Referring to the figures, exemplary embodiments of the invention will now be described. The following description will focus on the presently preferred embodiment of the present invention, which is implemented in desktop and/or server software (e.g., driver, application, or the like) operating in an Internet-connected environment running under an operating system, such as the Microsoft Windows® operating system. The present invention, however, is not limited to any one particular application or any particular environment. Instead, those skilled in the art will find that the system and methods of the present invention may be advantageously embodied on a variety of different platforms, including Macintosh®, Linux®, Solaris™, UNIX®, FreeBSD®, and the like. Further, implementation of the present invention does not require Internet connectivity. Therefore, the description of the exemplary embodiments that follows is for purposes of illustration and not limitation. The exemplary embodiments are primarily described with reference to block diagrams or flowcharts. As to the flowcharts, each block within the flowcharts represents both a method step and an apparatus element for performing the method step. Depending upon the implementation, the corresponding apparatus element may be configured in hardware, software, firmware or combinations thereof.

Please replace paragraph [0037] with the following rewritten paragraph:

CPU 101 comprises a processor of the Intel Pentium® family of microprocessors. However, any other suitable processor may be utilized for implementing the present invention. The CPU 101 communicates with other components of the system via a bi-directional system bus (including any necessary input/output (I/O) controller circuitry and other "glue" logic). The bus, which includes address lines for addressing system memory, provides data transfer between and among the various components. Description of Pentium-class microprocessors and their instruction set, bus architecture, and control lines is available from Intel Corporation of Santa Clara, CA. Random-access memory 102 serves as the working memory for the CPU 101. In a typical configuration, RAM of sixty-four megabytes or more is employed. More or less memory may be used without departing from the scope of the present invention. The read-only memory (ROM) 103 contains the basic input/output system code (BIOS) -- a set of low-level routines in the ROM that application programs and the operating systems can use to interact with the hardware, including reading characters from the keyboard, outputting characters to printers, and so forth.

Please replace paragraph [0040] with the following rewritten paragraph:

The computer system 100 displays text and/or graphic images and other data on the display device 105. The video adapter 104, which is interposed between the display 105 and the system's bus, drives the display device 105. The video adapter 104, which includes video memory accessible to the CPU 101, provides circuitry that converts pixel data stored in the video memory to a raster signal suitable for use by a cathode ray tube (CRT) raster or liquid crystal display (LCD) monitor. A hard copy of the displayed information, or other information within the system 100, may be obtained from the printer 107, or other output device. Printer 107 may include, for instance, an HP LaserJet® printer (available from Hewlett-Packard of Palo Alto, CA), for creating hard copy images of output of the system.

Please replace paragraph [0041] with the following rewritten paragraph:

The system itself communicates with other devices (e.g., other computers) via the network interface card (NIC) 111 connected to a network (e.g., Ethernet network, Bluetooth® wireless network, or the like), and/or modem 112 (e.g., 56K baud, ISDN, DSL, or cable modem), examples of which are available from 3Com of Santa Clara, CA. The system 100 may also communicate with local occasionally-connected devices (e.g., serial cable-linked devices) via the communication (COMM) interface 110, which may include a RS-232 serial port, a Universal Serial Bus (USB) interface, or the like. Devices that will be commonly connected locally to the interface 110 include laptop computers, handheld organizers, digital cameras, and the like.

Please replace paragraph [0042] with the following rewritten paragraph:

IBM-compatible personal computers and server computers are available from a variety of vendors. Representative vendors include Dell Computers of Round Rock, TX, Hewlett-Packard of Palo Alto, CA, and IBM of Armonk, NY. Other suitable computers include Apple-compatible computers (e.g., Macintosh®), which are available from Apple Computer of Cupertino, CA, and Sun Solaris™ workstations, which are available from Sun Microsystems of Mountain View, CA.

Please replace paragraph [0045] with the following rewritten paragraph:

Software system 200 includes a graphical user interface (GUI) 215, for receiving user commands and data in a graphical (e.g., "point-and-click") fashion. These inputs, in turn, may be acted upon by the system 100 in accordance with instructions from operating system 210, and/or client application module(s) 201. The GUI 215 also serves to display the results of operation from the OS 210 and application(s) 201, whereupon the user may supply additional inputs or terminate the session. Typically, the OS 210 operates in conjunction with device drivers 220 (e.g., "Winsock" driver -- Windows' implementation of a TCP/IP stack) and the system BIOS microcode 230 (i.e., ROM-based microcode), particularly when interfacing with peripheral devices. OS 210 can be provided by a conventional operating system, such as Microsoft Windows® 9x, Microsoft Windows® NT, Microsoft Windows® 2000, or Microsoft Windows® XP, all available from Microsoft Corporation of Redmond, WA. Alternatively, OS 210 can also be an alternative operating system, such as the previously mentioned operating systems.

Please replace paragraph [0146] with the following rewritten paragraph:

Consider the ~~following~~ sample application included in the source code appendix filed herewith.~~:~~

Please delete paragraph [0147].

Please replace paragraph [0148] with the following rewritten paragraph:

Given the code in the above-described sample application, the system of the present invention can recreate a full UML model at run time. At the outset, the packages making up the model for an application are compiled. This is done by an artificial model class that has the packages as attributes:

Please replace paragraph [0163] with the following rewritten paragraph:

The code generator 410 generates suitable code from operations performed on a design surface, and the compiler 420~~410~~ generates binary code from the source code emitted from the code generator. Those may be implemented using conventional code generator and compiler modules from an existing development environment (e.g., Borland C++, Borland JBuilder®, Borland Delphi, and so forth). The reflection reader 430 is a new core module which collaborates with the code generator and compiler modules to provide the below-described methodologies of the present invention. The reflection reader 430 is essentially fed a top-level node representing "the model". With that input, the reflection reader 430 may span the entire model using a tree spanning algorithm. During this process, the information from the UML model (found in code structures and decorations or "attributes") is reconstructed to represent the meta model 440. In essence, the meta model 440 holds the model information as generated by the reflection reader 430. This model information is, in turn, fed to a run-time framework that processes the model (represented at run-time as "RTModel" 450). The run-time model is a cached/speed optimized version of the meta model 440, which is then used during application execution.

Please replace paragraph [0166] with the following rewritten paragraph:

At code generation time (whether automatic, manual or semi-automatic) the information about a particular model element (e.g., a class, attribute, method, or the like) is added to the code element that the~~that~~ model element represents in the form of custom attributes. At run-time, when the model information is required (or at some appropriate time before that) the reflection reader reconstructs the model. Here, the reflection reader recreates for each code element (traversed using reflection) the corresponding appropriate model element. This mapping between code elements and model elements is sometimes straightforward, such as when a code class corresponds to a class in the model. At other times, however the mapping is more complicated. A property in code may be either an attribute in the model, or an association end, identified by certain custom attributes. Therefore, more complicated mappings may result, such as when a code class is identified as corresponding to a package in the model through the custom attribute [UmlMetaclass("package")].

Please replace the section heading of the claims (i.e., "Claims" which was automatically generated by the PTO Electronic Stylesheet v. 1.1.1, pursuant to electronic filing), with the following rewritten heading:

~~Claims~~What is claimed is: